INRIA

UNITÉ DE RECHERCHE
INRIA-RENNES

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
BP 105
78153 Le Chesnay Cedex
France
Tél. (1) 39 63 55 11

Rapports de Recherche

N° 1177

Programme 3
Réseaux et Systèmes Répartis

BUILDING A GLOBAL TIME ON PARALLEL MACHINES

Jean-Marc JEZEQUEL

Mars 1990

RR-1177

IRISA

INSTITUT DE RECHERCHE EN INFORMATIQUE
ET SYSTÈMES ALÉATOIRES

Campus Universitaire de Beaulieu
35042-RENNES CÉDEX
FRANCE
Téléphone: 99 36 20 00
Télex: UNIRISA 950 473 F
Télécopie: 99 38 38 32

Publication Interne n° 513
Février 1990 - 28 Pages

Building a Global Time on Parallel Machines

Jean-Marc JEZEQUEL
E-mail: jezequel@irisa.fr

Abstract

This paper presents a pragmatic algorithm to build a global time on any distributed system, which is optimal for homogeneous parallel machines. After some reflexions on time, clocks and distributed systems, we survey and criticize the classical approaches based on clock synchronisation techniques. Satisfying better our purposes, a statistical method is chosen as a building block to derive an original algorithm valid for any topology. This algorithm is particularly well suited for distributed algorithm experimentation purposes because, after an acquisition phasis, it induces neither CPU nor message overhead. We provide in the conclusion some data about its behavior and performances on some parallel machines.

Construction d'un temps global pour machines parallèles

Résumé

Ce rapport présente un algorithme pour construire de manière pragmatique une référence de temps global pour tout système réparti, et en particulier pour les machines parallèles homogènes. Après quelques remarques sur le temps, les horloges et les systèmes répartis, nous présentons et critiquons les approches classiques basées sur des techniques de synchronisation d'horloges. Pour répondre aux critères qu'on s'était fixés, nous avons choisi une méthode statistique comme brique de base pour dériver un algorithme original, valide pour toute topologie. Celui-ci est particulièrement bien adapté au contexte de l'expérimentation d'algorithmes distribués, car après une phase d'acquisition, il n'induit aucune surcharge, ni en temps UC ni en messages échangés. Nous donnons en conclusion quelques mesures de performances pour un certain nombre de machines parallèles.

CENTRE NATIONAL DE LA RECHERCHE SCIENTIFIQUE
(L. A. 227)
UNIVERSITÉ DE RENNES 1 I. N. S. A. DE RENNES

INSTITUT NATIONAL DE RECHERCHE
EN INFORMATIQUE ET EN AUTOMATIQUE
(LABORATOIRE DE RENNES)

Contents

List of Figures

1 Introduction

The concept of time is mainly subjective. However, ordering of events and duration between events seem to be objective basic time-related notions. But whereas ordering two events occuring at the same place is straightforward, there is some problem if the events occur at different places, or if we want to compare various durations, because usually there is no common time reference among them.

In a distributed system, a common time reference (*i.e.* a global time) is very useful for two kinds of reasons.

First, a global time availability allows to design simpler distributed algorithms to deal with synchronized behaviors, real-time constraints (like timeout for protocols) or actual ordering of events (Distributed Database Systems, version management...).

Then, if we want to observe the behavior of a distributed algorithm on a distributed system (for test or debug or other purposes), a global time allows us to measure its performances, to observe the order of events, and to verify easily some properties (mutual exclusion...).

This paper discusses the way such a global time may be actually constructed on parallel machines. We define in section 2 some vocabulary and precise what kind of systems we are interested in. Section 3 is an overview of the software clock synchronization principles, methods and limits. To get rid of those limits (too binding for our purposes), we present in section 4 a new hypothesis to justify an approach based on statistical estimations upon mutual dependencies of local clocks. Section 5 shows how this approach can be used as a building block to construct a global time in any distributed system, and that for some of them (the subclass of homogeneous parallel machines) this global time has the real time accuracy. We conclude with some practical results for this global time for some parallel machine, and on the interests and limits of this new service.

2 General Framework

2.1 About Time

In order to speak about time in an objective way, we have to define properly a small set of words. First, we are only interested in a Newtonian space, *i.e.* all the considered objects are supposed motionless in a same galilean reference. Nor are we interested in the actual nature of time: for us, time goes continuously and uniformly from past to future, so there is an isomorphism between time and the set $\Re$ of real numbers.

Agreeing with [12] we can now define an *event* as a *point* on the time line, and a *duration* as the interval between two events. The real number $t(e)$ is associated to event e by the fonction *time*. We call *clock* any abstract device which is able to measure durations. A *physical clock* is a device which can count the occurrence of quasi-periodic events. Such events are generally the observable oscillations of some physical system where the variations of a state variable of the system (position, volume, electrical tension...) are related to time through a periodic physical law. The motion of earth, a pendulum, a quartz or an atom are good examples of physical clocks. The *granularity* of a physical clock is the duration g between two incrementations of the clock. Two events e_1 and e_2 are said *concurrent* relatively to a physical clock if $\mid t(e_1) - t(e_2) \mid < g$, *i.e.* if it is not always possible to determine which one occurred first[1].

The *local time* lt_i is the continuous time generated by a physical clock C_i, taking as time basis its average granularity g_i. We can say that $lt_i = g_i(C_i + e_i)$, e_i being the "reading error" of the discrete clock C_i. If the reading of the physical clock C_i is independant from the C_i incrementation event, then $g_i.e_i$ is a random variable whose distribution is uniform on $[-g_i/2, g_i/2]$.

2.2 Distributed systems

In the following we consider that a distributed system is a set of processors (or sites, machines, nodes) communicating only by messages transmission through a point to point communication network. We call *parallel machine*

[1]It appears clearly that with this definition, the concurrency relation is *not* transitive.

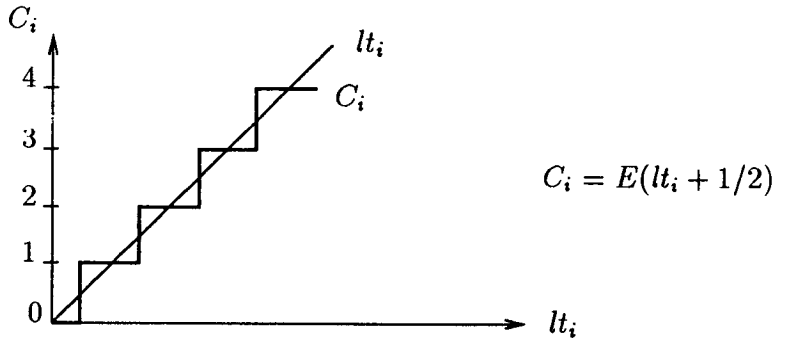

Figure 1: Local time from local clock

any homogeneous distributed system built on a local network. The transmission delays of messages are *not* negligible in front of internal action durations on a processor.

From [1] we know that there is no instantaneous global state of a distributed system. More precisely, it is impossible to observe such a global state from within the system. But there is no magical mean to prevent any external observer O (and why not a computer program) spying the system S with some extra hardware. For instance, if you look at the little LEDs on the front panel of an Intel iPSC hypercube, you can figure out what is the instantaneous global state of the machine. But either this observer O is considered outside of the distributed system, either the new system (S+O) is no longer a distributed system. So the borderline between a distributed system and the external world has to be very precisely defined. The well known example of [15] may illustrate this purpose: Let A,B,C be a computer network and X,Y external objects. X submits a request to A for C (request r_a), and then to Y for C through B (request r_b). From within the distributed system, there is no way to decide which one of r_a and r_b was first, whereas from outside, X can know that r_a was before. We call *external causality* (hereafter: $\rightsquigarrow$) the relation linking r_a to r_b, by reference to the *internal causality* (hereafter: $\rightarrow$) which exists for instance between the emission and the reception of a message.

Massively parallel machines like hypercubes or local networks of

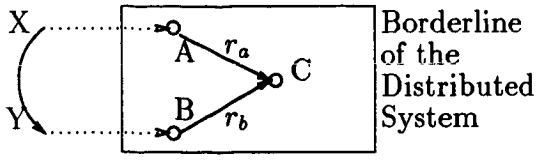

Figure 2: External causality vs. Internal causality

workstations are good examples of distributed systems. In order to compare different synchronization algorithms on those machines, let us present some typical data about some parallel machines:

- a network of Sun workstations, located on various buildings, and linked with *ethernet* (and optical couplers between buildings)
- Intel hypercube iPSC/2, with 64 processors (80386) linked by special hardware
- FPS hypercube T-40, with 32 processors (Transputers).

Measures have been performed with the ECHIDNA system, which provides an homogeneous interface for an high level programming language (Estelle) on parallel machines (see [10] for a presentation).

Machines	Sun	iPSC/2	FPS-T40
minimum transmission delay of a message T_{min} (ms)	10	1	3
maximum transmission delay of messages T_{max} (ms)	> 100	> 10	> 20
granularity of the available physical clock g (ms)	20	1	0.064
physical clock medium drift d (s/day)	0	≈ 1	≈ 0.5

2.3 Physical clock for common processors

Let us look into the usual way physical clock are built on common computers. There are two basic methods, the simpler (used in our Sun network) uses the 110 or 220V power line to cause an interrupt on every voltage cycle (at 50 or 60 Hz). The other one needs a local oscillator.

Excepting some special cases (cesium-beam oscillators for spatial or military applications), the most common oscillators are quartz crystals because they are simple, reliable, quite stable and above all, cheap.

The cut of the crystal determines its resonant frequency, within 50.10^{-6} of its nominal frequency for common commercial purposes (1.10^{-6} for military applications). This resonant frequency depends on the temperature and on few others factors of lesser importance. When such a crystal is mounted under tension, it generate a periodic signal which is fed into a counter to make it count down to zero. There, it causes a CPU interrupt (called *clock tick*): the CPU increments an internal register (its local "physical clock") and loads again the counter with the accurate predefined value.

According to [6] and to various experiments made on our parallel machines, if the temperature is quite constant at each node then the trajectory of a local time generated by such a physical clock may be modeled with a constant frequency offset: as the frequency change rate is less than $10^{-7}/day$, the resulting bias error on the time offset will be less than 30 ns for 10 min. So, we can model the trajectory of a local time generated by such a physical clock with:

$$lt(t) = \alpha + \beta t + \delta(t)$$

where:
α is the time offset at $t = 0$
β is the drift of the logical clock, $\beta = 1 + \kappa$, $\kappa = \frac{\Delta F}{F}$ (frequency offset)
$\delta(t)$ modelizes random perturbations and granularity

2.4 Global time and logical clock

On each site of a distributed system there is such a physical clock showing a different time. Our goal is to build on each site a *logical clock* such that all logical clocks show the same hour at the same time.

We call *global logical clock (LC)* an application from $\Re$ to $\Re^n$, such that:

$LC_i(t)$ is the value of the i^{th} component of LC at time t
Increasing $\forall i \in [1..n], \forall t \in \Re, \forall d > 0 \quad LC_i(t+d) - LC_i(t) \geq 0$
Agreement $\exists \epsilon \in \Re^+, \quad \forall i, j \in [1..n], \forall t \in \Re \quad \mid LC_i(t) - LC_j(t) \mid < \epsilon$

We call *global time* the time T generated on each site by the component LC_i. The imprecision of the global time is ϵ, and thus $G = 2\epsilon$ is its granularity.

As the major interest of a clock is to measure durations and to allow ordering of events, we are interested in the following properties for our logical clocks:

Accuracy $\exists \rho \in \Re, \forall t_1, t_2 \in \Re \times \Re, \forall i \in [1..n]$

$$(t_2 - t_1)(1 - \rho) < LC_i(t_2) - LC_i(t_1) < (t_2 - t_1)(1 + \rho)$$

i.e. the logical clock is within a linear envelope of real time. If LC_i is derivable, this property is equivalent to $\mid \frac{dLC_i(t)}{dt} - 1 \mid < \rho$.

Internal causality $\forall e_i, e_j$ internal events of a distributed system

$$e_i \rightarrow e_j \Rightarrow LC_i(t(e_i)) < LC_j(t(e_j))$$

External causality $\forall e_i, e_j$ observable events

$$e_i \rightsquigarrow e_j \Rightarrow LC_i(t(e_i)) < LC_j(t(e_j))$$

There exist known hardware solutions to build systems having such a global time, using phaselock loops or satellite synchronization. Besides the fact that such machines are no longer distributed systems, those solutions are quite expensive and not currently available for common parallel machines. So we have to check for software solutions, which have aroused a profuse bibliography.

3 Software Clock Synchronization

3.1 Principles and Problems

The usual way to synchronize two clocks C_i and C_j consists in choosing one of them (say C_i) as a reference (*i.e.* $\forall t$, $LC_i(t) = C_i(t)$), and then to measure the offset Δ_{ij} between lt_i and lt_j in order to set $LC_j(t) = C_j(t) - \Delta_{ij}$.

This approach rises the following problems:

PB1 As clocks have a granularity, we can only measure $\Delta_{ij} = C_j - C_i$ in place of $lt_j - lt_i$

PB2 As a frequency offset may exist between clocks, lt_i and lt_j can drift, thus Δ_{ij} is time dependent

Furthermore, in distributed systems the evaluation of Δ_{ij} is not trivial, because any information interchange can only be done through messages: site P_i has to send to site P_j the value at time t of its clock, C_i(t). So we have to deal with the following additional problem:

PB3 As the transmission time tm_{ij} of a message from P_i to P_j can't be known *a priori*, the error on the evaluation of Δ_{ij} can be as large as the maximum transmission time of a message.

It appears clearly that this method can be used only if transmission delays, granularities and frequency offsets are small enough with respect to the wanted agreement on the global time. Otherwise, we have to look for more sophisticated algorithms.

3.2 Lamport's logical clocks

Assuming the following hypothesis,

LH1 For every processor P_i having available a physical clock C_i, $\mid \frac{dC_i(t)}{dt} - 1 \mid < \kappa$
LH2 there exists a bound on the transmission delays of messages: $t_{del} = \mu + \xi$, where μ is the minimum delay of a message, and ξ the known bound on the unpredictable delay of messages.
LH3 every τ seconds a message is sent over every edge of the network
LH4 physical clock granularity is thin enough to timestamp two events of the same site with different values

Lamport presents in [15] an algorithm to build synchronous logical clocks, which can be adapted to synchronize physical clocks with the following methods:

- *as long as P_i doesn't receive any message, let* $\frac{dLC_i(t)}{dt} \equiv \frac{dC_i(t)}{dt}$
- *upon sending a message, P_j timestamps it with the value* $T_j = LC_j(t)$
- *when P_i receives a message, let* $LC_i(t) = max(LC_i(t), T_j + \mu)$

The properties of those logical clocks are:

LP1 $\forall i, j \; \forall t \mid LC_i(t) - LC_j(t) \mid < \epsilon \approx 2\kappa\tau + \xi$ which is for our examples: $\epsilon_{Sun} > 50ms, \epsilon_{iPSC} > 10ms, \epsilon_{FPS} > 20ms$
LP2 $\mid \frac{dLC_i(t)}{dt} - 1 \mid < \kappa$ between resets, but undefined during resets.
LP3 By construction, internal causality is respected

But we can remark that the resulting global time doesn't stay within a linear envelope of real time, because clocks are always reset forward (see figure 2), each τ in the bad case. On the other hand, if τ is large then ϵ is large, and thus the agreement poor.

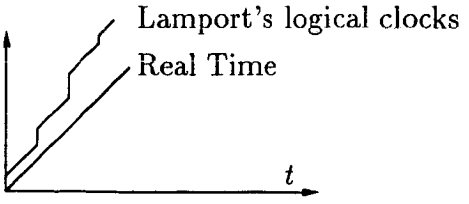

Figure 3: Lamport's logical clocks

Anyway, if the application doesn't send enough messages to complete LH3, additional messages are needed, and thus the application can be perturbed.

Furthermore, granularity is not really taken into account, meanwhile in our systems LH4 is not valid (excepted for the FPS).

3.3 Improvements and limits

Numerous researchers improve the Lamport's idea: among them, Lamport himself in [17] (where clocks are no longer always reset forward, and can tolerate some faults), and Marzullo who formalizes the problem and its solution in [19, 20]. Various algorithms are presented to deal with byzantine behaviors in [16, 18, 9] and in [24](optimal solution with respect to the Accuracy property: its global time accuracy is as good as the harware one (quartz)); see [22] and [23] for an overview and a detailed comparison.

But, as it is highlighted in [13, 14], those algorithms are quite complex, and thus difficult to implement correctly and maintain, and exhibit high CPU and messages overheads as the number of tolerated faults increases. Furthermore, it is observed in [3] that the overall increase in reliability provided by those byzantine algorithms is not always significant, compared to other sources of system failure. Thus, restricting the class of faults to be resisted, [3] presents a simpler algorithm, which improves the precision of the global time (the trick is to lower $t_{del} = \mu + \xi$ (and thus ϵ), and to consider that a message whose transfer delay is greater than t_{del} is faulty).

So, apart for the accuracy and the fault tolerance problems, the primary solution of [15] has not been drastically improved. When extra messages are

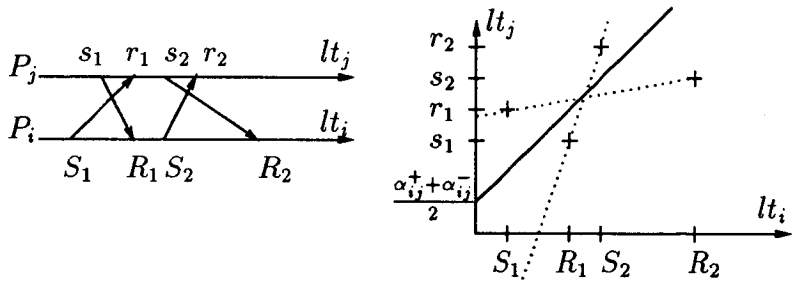

Figure 4: Statistical estimation of time and frequency offset

exchanged to control the mutual drift of physical clocks, the observational purpose of a global time is clearly unusable: those extra messages perturb the application that we want to observe. Furthermore, the precision of the global time is still not very good, because it depends closely on the variability of the transmission delays of messages, which is always the main factor of uncertainty.

But, according to hypothesis LH1, LH2, LH3, [4] shows that much better results are impossible. So, if we want to go further, we have to study new hypothesis, reformulate the problem and present new methods.

4 Statistical Estimation of Clock Offsets

4.1 Elimination of the transmission delay uncertainty

In a first step, we are not interested in fault tolerance (in parallel machines, almost all faults are software bugs or fail stops). Furthermore, we think that clock synchronization is not an intrinsic problem of fault tolerance. As it is explicitely stated in [3], the major contribution brought by thoses techniques is to get rid of the transmission delay uncertainty (and eventually to deal with crash and join problems).

But transmission delays can be modelized by a random variable whose distribution is unknown, because they depend on the software overhead to access the network on the sending machine, on the network transmission

time (depending on the size of the messages) and on the software overhead to deliver the incoming messages to the right tasks on the receiving machine, etc...

Reference [7] proposes a statistical method to eliminate this uncertainty when drift between clocks may be assumed negligible for short periods. Instead of performing only one message exchange when a resynchronization is needed, the algorithm performs numerous exchanges and selects the one with the best transmission delay to compute the time offset between two sites (and the precision of this evaluation).

The major advantage of this algorithm is its precision obtained on the global time, which is only limitated by the granularity of the physical clocks and the anisotropy of the network (*i.e.* difference between minimum transmission delays). Furthermore the CPU overhead is reasonably low.

However, if the mutual drift between physical clocks is not negligible, frequent resynchronizations will be needed, and numerous messages added to the application: this global time is no longer suitable for observation purposes.

Until now we supposed that a bound was known on the mutual drift between physical clocks. According to this bound, a resynchronization round was triggered to compute new time offsets when the possible uncertainty due to mutual drifts became too high. The only way to avoid those resynchronization rounds would be to actually *compute* the frequency offset between physical clocks.

4.2 Frequency Offset Evaluation

The idea of [5] is also to use multiple message exchanges between P_i and P_j to estimate with a statistical method the time *and* the frequency offset between two sites. As for each site P_i we have $lt_i(t) = \alpha_i + \beta_i t + \delta_i$ (see 2.3), there exist α_{ij} and β_{ij} such that $lt_j(t) = \alpha_{ij} + \beta_{ij} lt_i(t) + \beta_{ij}\delta_{ij}$, where δ_{ij} is a random variable whose density function is the convolution between δ_j and δ_i density functions ($\beta_{ij} = \beta_j/\beta_i \;\; \alpha_{ij} = \alpha_j - \beta_{ij}\alpha_i \;\; and \;\; \delta_{ij} = \delta_j - \beta_{ij}\delta_i$). The actual purpose of [5] is thus to compute bounds for α_{ij} and β_{ij}.

Let S_i^k be the event of site P_i sending a message to P_j, R_j^k the corresponding reception on P_j, and τ_{ij} the positive random variable (whose distribution is unknown) modelizing the transmission delay of messages.

Thus, we can write for messages received by P_j (see figure 3):

$$lt_j(t(R_j^k)) = lt_j(t(S_i^k)+\tau_{ij}) = lt_j(t(S_i^k))+\beta_j\tau_{ij} = \alpha_{ij}+\beta_{ij}lt_i(t(S_i^k))+\beta_j\tau_{ij}+\delta_{ij}$$

and symetricaly for messages sent by P_j:

$$lt_j(t(S_j^k)) = lt_j(t(R_i^k)-\tau_{ji}) = lt_j(t(R_i^k))-\beta_j\tau_{ji} = \alpha_{ij}+\beta_{ij}lt_i(t(R_i^k))-\beta_j\tau_{ji}+\delta_{ij}$$

If δ_{ij} (mainly the granularity) is small in comparison with transmission delays ($\delta_{ij} \ll \tau_{ij}$ so $\beta_j\tau_{ij} + \delta_{ij} > 0$ and $-\beta_j\tau_{ji} + \delta_{ij} < 0$), then the wanted line ($lt_j = \alpha_{ij} + \beta_{ij}lt_i$) geometrically lies between the two separate sets of points UP=(S_i^k,R_j^k) and LOW=(R_i^k,S_j^k); see figure 3. Thus, we can compute higher and lower bounds α_{ij}^+,α_{ij}^- and β_{ij}^+,β_{ij}^- for α_{ij} and β_{ij} using a geometrical algorithm[2]. Furthermore, as the duration and the number of points of the estimation increase, $\beta_{ij}^+ - \beta_{ij}^-$ decreases to 0, and $\alpha_{ij}^+ - \alpha_{ij}^-$ to the difference between minimum transmission delays in the two directions. Practically, this convergence is very fast for distributed systems built on local networks, because the transmission delay distribution has an important mass near the minimum (see [8] for details and proof).

Hence, the problem of transmission delay variability is solved, and as the mutual frequency offset is computed instead of being bounded, resynchronization is no longer needed. This last property is very interesting because, after an acquisition phasis, this algorithm induces neither CPU nor message overhead. The application is thus no longer perturbed by the global time construction.

4.3 Optimisations and limits for parallel machines

Let μ be a constant less than the minimum transmission delay (more precisely, μ is such that $\forall i,j \;\; 0 < \mu < \beta_j\tau_{ij}$), and δ such that $\forall i,j \;\; | \delta_{ij} | < \delta$. $\delta \approx g$ (the granularity) because $| \delta_{ij} | = | \delta_i \otimes \delta_j | < 2\max(| \delta_i |)$, and $| \delta_i | \approx g/2$ is mainly the precision of the physical clock ($\otimes$ is the convolution product).

A problem arises when the granularity is not small with respect to transmission delays ($\delta > \mu$, see the Sun network example): the two sets of points (S_i^k,R_j^k) and (R_i^k,S_j^k) may overlap, because $\beta_j\tau_{ij} + \delta_{ij}$ is no longer always greater than $-\beta_j\tau_{ji} + \delta_{ji}$.

[2]The first idea was to use a linear regression to estimate α_{ij} and β_{ij}, but [5] shows that this method can't give 100% confidence intervals.

The solution to this problem consists in artificially increasing the minimum transmission delay so that $\mu' > \delta$. Geometrically, this leads to move away the two sets of points each other. The required minimum value of this adjustment is thus $\nu = \delta - \mu$, so that $\nu + \beta_j \tau_{ij} + \delta_{ij} > -\nu - \beta_j \tau_{ji} + \delta_{ji}$.

We can use the same method to decrease the value of the minimum transmission delay to $\mid \mu_{ij} - \mu_{ji} \mid$ (*i.e.* the difference between minimum transmission delays in both directions) as long as the two sets of points (S_i^k, R_j^k) and (R_i^k, S_j^k) remain separated each other, *i.e.* the adjustment[3] ν is such that $\nu > \delta - \mu$.

This can be used to gain precision on the evaluation of β_{ij} and α_{ij} with the same acquisition period ΔT, as precision on α_{ij} depends on the precision on β_{ij} (and is limited by $\mid \mu_{ij} - \mu_{ji} \mid$), and the lower bound for $\Delta\beta_{ij} = \frac{1}{2}(\beta_{ij}^+ - \beta_{ij}^-)$ is $\frac{2\mu}{\Delta T}$.

If the network is highly symmetrical and homogeneous (as it is in our three examples), minimum transmission delays in both directions can be considered equal (isotropic networks), and thus lower bound[4] for $\Delta\beta_{ij}$ is $\frac{2(\mu+\nu)}{\Delta T} = \frac{2\delta}{\Delta T}$.

This bound is however surestimated, as $\delta > \mid \delta_{ij} \mid = \mid \delta_i \otimes \delta_j \mid$, where δ_i, δ_j are two independant random variables, whose distribution is uniform on $[-g/2, g/2]$ (if we can neglect the time offset rate factor). The calculus shows that this lower bound has a probability of 75% to be smaller than $\frac{g}{\Delta T}$.

5 Building a global time over any connected network

5.1 Theoretical solution

At this point, a site is able to evaluate its local linear dependencies with its neighbours. From those local dependencies, we want to derive some global dependencies in order to build on each processor a global time function $TG_i(lt_i(t))$ having the required properties.

We assume in this part that each processor P_i of the network N has computed (in parallel) its linear dependencies with all of its neighbours with

[3] If $\nu > 0$ the sets are moved away and else they are bring together

[4] Actual value is computed dynamically.

the algorithm described in the previous part (let us call it A1). So, the following system (S) of inequations holds:

$$\forall P_i \in N,\ \forall P_j \in V_i,\ \alpha_{ij}^- < \alpha_{ij} < \alpha_{ij}^+,\ \ \beta_{ij}^- < \beta_{ij} < \beta_{ij}^+$$

and

$$lt_j(t) = \alpha_{ij} + \beta_{ij} lt_i(t) + \delta_{ij}$$

where V_i denotes the set of P_i's neighbours. As $\mid \delta_{ij} \mid \approx g$, we have:

$$\forall P_i \in N,\ \forall P_j \in V_i,\ \alpha_{ij}^- - g + \beta_{ij}^- lt_i(t) < lt_j(t) < \alpha_{ij}^+ + g + \beta_{ij}^+ lt_i(t)$$

As $lt_i(t) = \alpha_i + \beta_i t + \delta_i$, and $\mid \delta_i \mid \approx g/2$, (S) is equivalent to

$$\alpha_{ij}^- - g + \beta_{ij}^-(\alpha_i - g/2) + \beta_{ij}^- \beta_i t < \alpha_j + \beta_j t < \alpha_{ij}^+ + g + \beta_{ij}^+(\alpha_i + g/2) + \beta_{ij}^+ \beta_i t$$

This is valid for each t, so:

$$\forall P_i \in N,\ \forall P_j \in V_i,\ \beta_{ij}^- \beta_i < \beta_j < \beta_{ij}^+ \beta_i$$

and

$$\alpha_{ij}^- - g + \beta_{ij}^-(\alpha_i - g/2) < \alpha_j < \alpha_{ij}^+ + g + \beta_{ij}^+(\alpha_i + g/2)$$

As the method used in A1 is highly symetric, $\beta_{ij}^+ = 1/\beta_{ji}^-$ and $\alpha_{ij}^+ = -\alpha_{ji}^-/\beta_{ij}^-$, so half of the inequations are redondant. We want to solve this system, *i.e.* to find intervals $[\alpha_i^-, \alpha_i^+]$ and $[\beta_i^-, \beta_i^+]$ for all P_i that verify the system. There exists at least a solution: if every site has a linear dependency with all of its neighbours and if the graph is connected, every site has a linear dependency with every other by composition of the dependencies along any path. There exists even an infinity of solutions, each one being homothetic of the others (in terms of β).

The principle of the solution is to choose a reference site P_r (either statically or by regular dynamical election), where we state $\beta_r^- = \beta_r^+ = 1$ and $\alpha_r^- = \alpha_r^+ = 0$; and then to eliminate *all* the inequations of (S) by substitution.

Let $d(i,j)$ be the topological distance from P_i to P_j on N. Let $D_r(p) = \{j \in N \mid d(r,j) = p\}$ be the set of sites which are at distance p from P_r. We suppose that there exists a minimal spanning tree T over N (*i.e.* where distance from P_r to P_j along T is $d(r,j)$), whose root is P_r and diameter d,

and that every process P_i knows its depth and its neighbour's one on this tree[5].

The system will be solved from near to near along T, *i.e.* for the neighbours of P_r, then for the neighbours of the neighbours etc...

We say that a site P_j is *synchronized* if and only if all the inequations of (S) with terms in α_{ij} or β_{ij} such that $d(r,i) \leq d(r,j)$ have been eliminated (so we have found the wanted intervals $[\alpha_j^-, \alpha_j^+]$ and $[\beta_j^-, \beta_j^+]$). Let be S^p the system where $\forall P_j \in \bigcup_{k\in[0,p]} D_r(k)$, P_j is synchronized.

Theorem *If P_r is the root of a minimal spanning tree T and if P_r is synchronized, then it is possible to eliminate all the inequations of (S) in order to synchronize all the sites of T in d steps, where d is the depth of T.*

Demonstration (by induction on the depth of the graph):

Initially, only site P_r is synchronized, and S^0=(S). Suppose that:

$$\forall P_j \in \bigcup_{k\in[0,p-1]} D_r(k),\ P_j \textit{ is synchronized}$$

The graph is synchronized until depth $p-1$, and remaining inequations form the system S^{p-1}. Let us see how to synchronize it at depth p.

Let be $P_j \in D_r(p)$ and $P_i \in V_j$. As $d(i,j) = 1$, $V_j \subset D_r(p-1) \cup D_r(p) \cup D_r(p+1)$. So we have to eliminate all the inequations of S^{p-1} having terms in α_{ij} or β_{ij} such that $P_i \in D_r(p-1) \cup D_r(p)$.

1. If $P_i \in D_r(p-1)$, then P_i is synchronized (by hypothesis), so there exist solutions such that $\alpha_i^- < \alpha_i < \alpha_i^+$ and $\beta_i^- < \beta_i < \beta_i^+$. In S^{p-1} we can extract the two following inequations[6]:

   $$\beta_{ij}^- \beta_i < \beta_j < \beta_{ij}^+ \beta_i \text{ and } \alpha_{ij}^- - g + \beta_{ij}^- (\alpha_i - g/2) < \alpha_j < \alpha_{ij}^+ + g + \beta_{ij}^+ (\alpha_i + g/2)$$

   This is valid for all $P_i \in V_j \cap D_r(p-1)$, so the conjonction of those inequations gives:

   $$[\beta_j^-, \beta_j^+] = \bigcap_{P_i \in V_j \cap D_r(p-1)} [\beta_i^- \beta_{ij}^-, \beta_i^+ \beta_{ij}^+]$$

[5] If every site can know statically the topology of R, then T can be statically defined. Otherwise T has to be built, using for example the algorithm (say A2) provided in [2], after whom each site broadcasts its depth in T to its neighbours.

[6] The symetric one (in terms of α_{ji} and β_{ji}) are equivalent to those one.

and

$$[\alpha_j^-, \alpha_j^+] = \bigcap_{P_i \in V_j \cap D_r(p-1)} [\alpha_i^- - g + \beta_{ij}^-(\alpha_i^- - g/2), \alpha_i^+ + g + \beta_{ij}^+(\alpha_i^+ + g/2)]$$

2. If $P_i \in D_r(p)$, let us notice that $D_r(p)$ forms a sub-network N_p, partitioned in a set of connected sub-networks $N_p^1..N_p^n$.

   - If $N_p^j = \{P_j\}$, *i.e.* if $V_j \cap D_r(p) = \emptyset$, then there is no more inequations over P_j, so P_j is synchronized.
   - Otherwise, for each of those connected sub-networks N_p^k we choose again (either statically or dynamicaly) a new reference site, root of a new minimal covering tree over N_p^k, and we synchronize those sub-networks with the same method as for the main network.

     As the number of sites in N_p^k is strictly less than the number of sites in N (the reference site of N can't be in N_p^k), this leads to build decreasing sequences (with the inclusion meaning) of unsynchronized sub-networks whose minimal size is one site. Upon synchronization of thoses last sites (a single site network is synchronized by definition), N_p^k becomes synchronized, thus $P_j \in N_p^j$ is synchronized.□

5.2 Algorithm derivation

From the mathematical resolution of (S), we can derive directly a distributed algorithm A3 to make the problem be solved by the considered distributed system.

$A3(N, P_r)$:= -- Algorithm to synchronize network N, with P_r as reference[7].

for each process P_j *do begin*
p:=distance(P_j, P_r)*;*
-- **STEP 1**, get the values from the already synchronized neighbours
if $p > 0$ *then begin*
$\forall P_i \in V_j \cap D_r(p-1)$ P_i ? $([\alpha_i^-, \alpha_i^+], [\beta_i^-, \beta_i^+])$;

[7]Hereafter, "P_i ? " denotes the asynchronous reception of a message from P_i, and "P_i !" the emission to P_i.

$[\beta_j^-, \beta_j^+] := \bigcap_{P_i \in V_j \cap D_r(p-1)} [\beta_i^- \beta_{ij}^-, \beta_i^+ \beta_{ij}^+]$;
$[\alpha_j^-, \alpha_j^+] := \bigcap_{P_i \in V_j \cap D_r(p-1)} [\alpha_i^- + \beta_{ij}^- \alpha_{ij}^-, \alpha_i^+ + \beta_{ij}^+ \alpha_{ij}^+]$;
end;
-- **STEP 2**
if $V_j \cap D_r(p) \neq \emptyset$ *then begin*
Perform A2(N_p^j, P_s)*;* -- to build a minimal covering tree (whose root is P_s)
-- over the subnetwork N_p^k (possible static knowledge)
Perform A3(N_p^j, P_s)*;* -- recursive call
end;
-- **STEP 3**, as P_j is synchronized, it broadcasts its values deeper on the tree
$\forall P_i \in V_j \cap D_r(p+1)$ P_i ! $([\alpha_j^-, \alpha_j^+], [\beta_j^-, \beta_j^+])$;
end;

So, the full algorithm is:

Perform A1;
All sites exchange their relatives values $\alpha_{ij}^+, \alpha_{ij}^-, \beta_{ij}^+, \beta_{ij}^-$ *with neighbours*
Perform A2(N, P_r)*;*
Perform A3(N, P_r)*;*
$LC_i(C_i(t)) ::= (C_i(t) - \frac{\alpha_i^+ + \alpha_i^-}{2}) / (\frac{\beta_i^+ + \beta_i^-}{2})$;

5.3 Application to some classical topologies

Our algorithm doesn't need to know statically the actual topology of the network. But if it is known, we can derive from it simpler algorithms to suit the particularities of the network, because the second step of the general algorithm (which could be quite costly) will be streched.

Fully connected network In such a network, the depth of the covering tree is always 1. So, all sites are chosen one after the other to be the next reference site for the remaining subnetwork. During the STEP 1 of A3, the only message expected is from the reference site of the current subnetwork, and the STEP 3 is performed only by the site which is reference of its subnetwork. Supposing that A2 is not actually performed (static order is known to choose next reference site), the time complexity of A3 is the complexity on the last site chosen: $O(n)$. We can notice that we get there a distributed version of the algorithm first presented in [5].

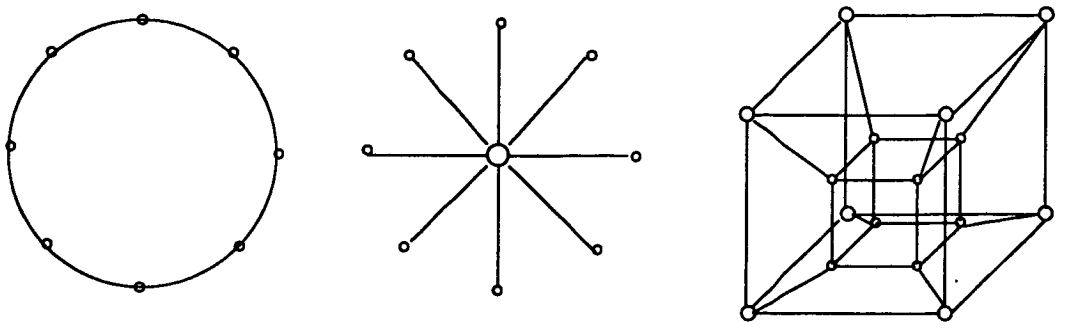

Figure 5: Ring, Star and Hypercube topologies

Ring topology Let be R_n a network of n processors connected with a ring topology. We must study two cases, depending on the parity of n:

- If n is even, $\forall P_j \in R_n$, $V_j \cap D_r(p) = \emptyset$. So, the second step of A3 becomes useless, and its time complexity is $O(n/2)$.
- If n is odd, the two sites located at distance $\frac{n-1}{2}$ of P_r are neighbours. So, the second step of A3 is usefull only for those two sites, and time complexity is also $O(\frac{n}{2})$.

Star topology If we choose the center of the network as the reference site, the minimal covering tree depth is only one, and thus A3 complexity becomes $O(1)$.

Hypercube topology Let be P_r a site of an hypercube H, whose dimension (and thus diameter) is d. Hypercube topologies have numerous interesting particularities. Among them, if $P_s \in D_r(p)$, then $V_s \cap D_r(p) = \emptyset$: a site which is at distance p from P_r has no neighbour at the same distance p from P_r.

So, in the same way that for an even ring, the second step of A3 becomes useless, and time complexity is $O(d)$.

5.4 Global time and real time

As we have chosen the physical clock on one site to synchronize all the others, the resulting global time can't be better (in terms of accuracy) than the local time generated by the quartz of the reference site. This is generally not a problem when we are only interested in internal events observation. However, if we have to deal with external events references, a synchronization with a better external clock might be required. If such a clock is connected to the network, it is possible to select its site as the primary reference for our algorithm, so the accuracy of the global time is its accuracy.

However in almost all parallel machines, such a good clock is not available. But on homogeneous networks (such as hypercube machines), the available quartz have a resonant frequency that can be modelized on the set of all the quartz of the network by a random variable whose mean value is the nominal frequency of the quartz (see [21]). So,

$$\frac{1}{N}\sum_{i=1}^{N}\beta_i = 1 \pm \varepsilon, \quad \varepsilon \approx 10^{-10}$$

Thus it is possible to append to our algorithm a last phase where it adjusts all the computed frequency offset with their mean value. We can then have an accuracy very close to real time, without any reference to a better than quartz (*e.g. atomic*) external clock.

6 Conclusion

The algorithm described above has been simplified for the hypercube topology and specified with the Estelle programming language in order to be compiled with ECHIDNA and experimented on Sun network, on iPSC/2 and on FPS-T40.

An initial acquisition period of 300 s yields to the following results:

Machines	Sun	iPSC/2	FPS-T40
Physical clock granularity g (ms)	20	1	0.064
Measured bound on $\Delta F/F$	0	2.10^{-5}	5.10^{-6}
Best precision with classical methods (ms)	80	10	20
Precision on β_i	6.10^{-5}	4.10^{-6}	5.10^{-7}
Initial precision of LC_i(t) (ms)	50	2	0.2
Precision after one hour (ms)	200	15	2

The hypothesis of linearity between physical clocks (*i.e.* that frequency offset rate is actually negligible) has been verified for rather large periods ($\approx 12h$) through various experiments[11]. However, the imprecision of our global time increases linearly with time: $G \approx 2g + 2\Delta\beta t \approx 2g + \frac{4\delta}{\Delta T}t$. So, if this algorithm is to be used continuously, resynchronization rounds should take place dynamically when the precision becomes too bad for the current purpose, or application messages can be used to enforce the precision on frequency mutual offsets during the application execution.

The granularity of our global time doesn't always allow to verify internal causality (for the Sun and the iPSC). But, as shown by the FPS example, this is mostly an hardware problem, and there is no software possibility to get rid of physical clock granularity. However, as stated in 4.3, this granularity is likely to be very over estimated. We have verified this fact with the following experiment.

Let a token travel as fast as possible on a ring embedded on the iPSC. Each site increments the value held by the token, and store the date of the various receptions (with both local and global times). This experiment has been repeated many times, and causality violation with the global time never occured.

So, practically our global time can be used to order (with high probability) communication events, even on the iPSC. But, if we would use this new service in the property verification field, it would be interesting to study the idea of a probabilistic order of events, in place of classical partial orders.

For the distributed algorithm experimentation purpose, this global time algorithm has been integrated in the ECHIDNA system (along with the Lamport's algorithm to ensure internal causality), so it is now possible to observe a distributed algorithm behavior on a parallel machine without perturbing it.

Acknowledgment

Special acknowledgment is due to Y. Haddad for the original idea of frequency offset estimation, to C. Jard for his constant support and advices, and to all the members of the ADP team for their reviewing.

References

[1] K. M. Chandy and L. Lamport. Distributed snapshots: determining global states of distributed systems. *ACM TOCS*, 63–75, February 1985.

[2] A. T. Cheung. Graph traversal techniques and the maximum flow problem in distributed computing. *IEEE Trans. on SE*, SE-9(4):504–512, 1983.

[3] F. Cristian, H. Aghili, and R. Strong. Clock synchronization in the presence of omission and performance faults, and processors joins. In *Proc. of 16th IEEE Symposium on Fault-Tolerant Computing Systems, Vienna*, pages 218–223, July 1986.

[4] D. Dolev, J.Y. Halpern, and R. Strong. On the possibility and impossibility of achieving clock synchronization. In *Proc. of 16th ACM Symposium on Theory of Computing*, pages 504–511, April 1984.

[5] A. Duda, G. Harrus, Y. Haddad, and G. Bernard. Estimating global time in distributed system. In *Proc. 7th Int. Conf. on Distributed Computing Systems, Berlin*, 1987.

[6] C.E Ellingson and R.J. Kulpinski. Dissemination of system time. *IEEE Transactions on Communications*, COM-21(5):605–623, May 1973.

[7] R. Gusella and S. Zatti. A network time controler for a distributed berkeley UNIX system. *IEEE Distr. Proc. Tech. Comm. Newsletter*, SI-2(6):7–15, June 1984.

[8] Y. Haddad. Performance dans les systèmes répartis: des outils pour lès mesures. Thèse de Doctorat, Univ. Paris-Sud, Centre Orsay, PARIS, Septembre 1988.

[9] J.Y. Halpern, B. Simons, R. Strong, and D. Dolev. Fault-tolerant clock synchronization. In *Proc. of the Third ACM Symposium on Principles of Distributed Computing, Vancouver, Canada*, pages 89–102, August 1984.

[10] C. Jard and J.-M. Jézéquel. A multi-processor Estelle to *C* compiler to experiment distributed algorithms on parallel machines. In *Proc. of the 9th IFIP International Workshop on Protocol Specification, Testing, and Verification, University of Twente, The Netherlands*, North Holland, 1989.

[11] C. Jard and J.-M. Jézéquel. Outils pour l'expérimentation d'algorithmes distribués sur machines parallèles. In *Actes du Colloque C^3 d'Angoulème*, GRECO C^3/CNRS, December 1988.

[12] H. Kopetz. Accuracy of time measurement in distributed real time systems. In *5th Symposium on Reliability in Distributed Software and Database Systems*, pages 35–41, IEEE Comp. Society, 1986.

[13] H. Kopetz and W. Ochsenreiter. Clock synchronization in distributed real time systems. In *IEEE Transaction on Computers, Special issue on Real Time Systems*, pages 933–940, August 1987.

[14] C.M. Krishna and K.G. Shin. Synchronization and fault-masking in redundant real time systems. In *Proc. of the FTCS 14, IEEE Press*, pages 151–157, 1984.

[15] L. Lamport. Time, clocks and the ordering of events in a distributed system. *Communications. of the ACM*, 21(7):558–565, July 1978.

[16] L. Lamport and P.M. Melliar-Smith. Byzantine clock synchronization. In *Proc. of the Third ACM Symposium on Principles of Distributed Computing, Vancouver, Canada*, pages 68–74, August 1984.

[17] L. Lamport and P.M. Melliar-Smith. Synchronizing clocks in the presence of faults. *Journal of the ACM*, 32(1):52–78, Juanary 1985.

[18] J. Lundelius and N. Lynch. A new fault-tolerant algorithm for clock synchronization. In *Proc. of the Third ACM Symposium on Principles of Distributed Computing, Vancouver, Canada*, pages 75–88, August 1984.

[19] K. Marzullo. Loosely coupled distributed services: a distributed time service. PhD Dissertation. Stantford University, Computer System Laboratory, 1983.

[20] K. Marzullo and S. Owiki. Maintaining time in a distributed system. In *ACM Operating Systems Rev.*, pages 44–54, 1983.

[21] R. Meyer. MOS crystal oscillator design. *IEEE journal of solid state circuits*, SC-15(2), April 1980.

[22] F.B. Schneider. A paradigm for reliable clock synchronization. In *Proc. Advanced Seminar Real Time Local Area Network*, pages 85–104, April 1986.

[23] B. Simons, J. Lundelius, and N. Lynch. *An Overview of Clock Synchronization.* Technical Report, IBM Research Division, October 1988.

[24] T.K. Srikanth and S. Toueg. Optimal clock synchronization. In *4th Annual ACM Symposium on Principles of Distributed Computing*, pages 71–86, August 1985.

LISTE DES DERNIERES PUBLICATIONS INTERNES

PI 506 **VM_pRAY, AN EFFICIENT RAY TRACING ALGORITHM ON DISTRIBUTED MEMORY PARALLEL COMPUTER**
Didier BADOUEL, Thierry PRIOL
Janvier 1990, 50 Pages.

PI 507 **ON THE REGULAR STRUCTURE OF PREFIX REWRITINGS**
Didier CAUCAL
Janvier 1990, 32 Pages.

PI 508 **RAY TRACING ON DISTRIBUTED MEMORY PARALLEL COMPUTERS: STRATEGIES FOR DISTRIBUTING COMPUTATIONS AND DATA.**
Didier BADOUEL, Kadi BOUATOUCH, Thierry PRIOL
Janvier 1990, 16 Pages.

PI 509 **STABILITY ANALYSIS AND IMPROVEMENT OF THE BLOCK GRAM-SCHMIDT ALGORITHM**
William JALBY, Bernard PHILIPPE
Janvier 1990, 24 Pages.

PI 510 **TESTING FOR THE UNBOUNDEDNESS OF FIFO CHANNELS IN PROGRAMS.**
Thierry JERON
Janvier 1990, 30 Pages.

PI 511 **AUTOMATIC ANIMATION CONTROL OF PHYSICAL SYSTEMS.**
Georges DUMONT, Bruno ARNALDI, Gérard HEGRON
Janvier 1990, 22 Pages.

PI 512 **A FAULT TOLERANT TIGHTLY COUPLED MULTIPROCESSOR ARCHITECTURE BASED ON STABLE TRANSACTIONAL MEMORY**
Michel BANATRE, Philippe JOUBERT
Février 1990, 20 Pages.

PI 513 **BUILDING A GLOBAL TIME ON PARALLEL MACHINES**
Jean-Marc JEZEQUEL
Février 1990, 28 Pages.

Imprimé en France
par
l'Institut National de Recherche en Informatique et en Automatique